ARTICLE REPRINT FROM
Electronics international
FEBRUARY 16, 1978

8086 microcomputer bridges the gap between 8- and 16-bit designs

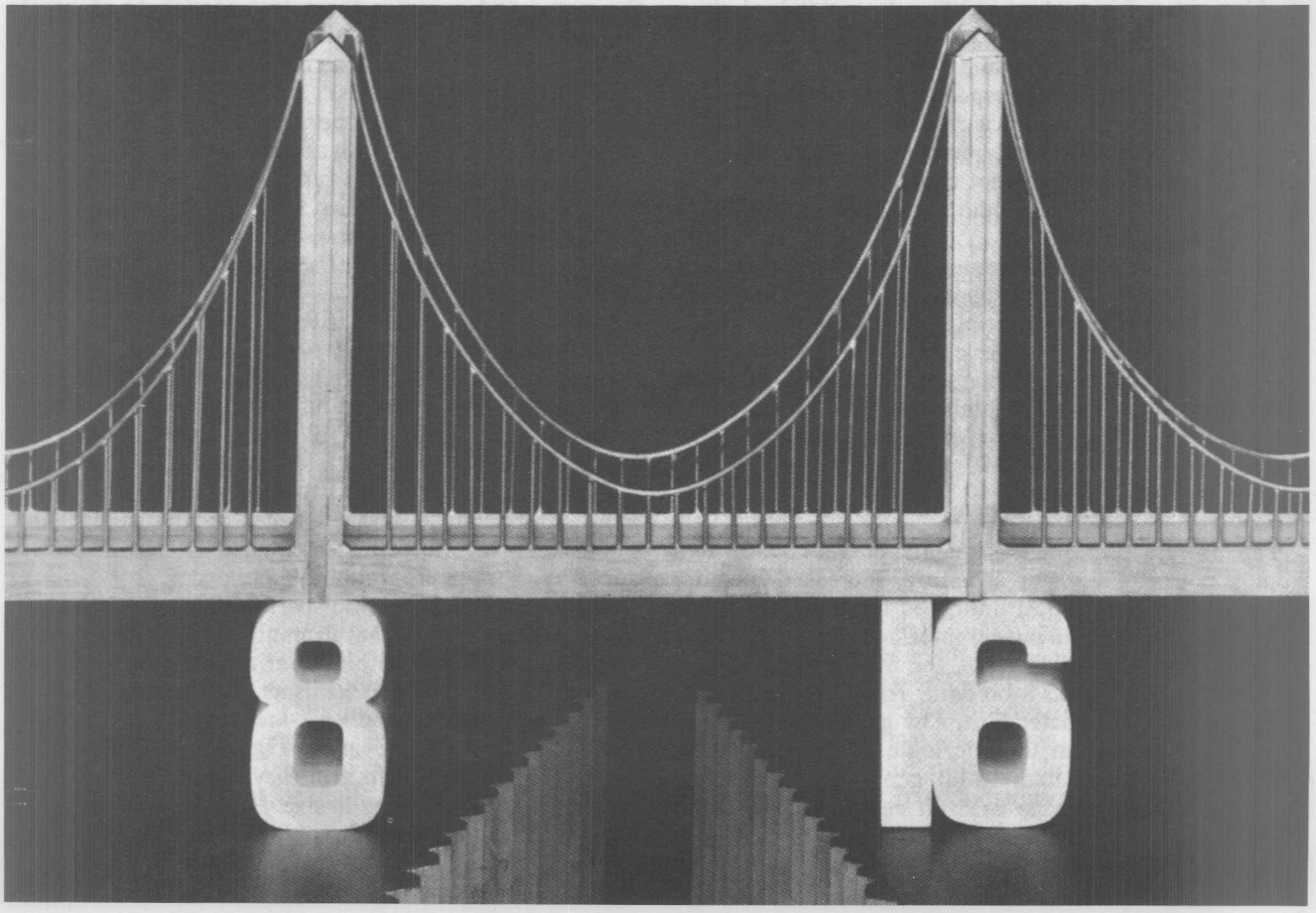
by B. Jeffrey Katz, Stephen P. Morse, William B. Pohlman, and Bruce W. Revenel, *Intel Corp., Santa Clara, Calif.*

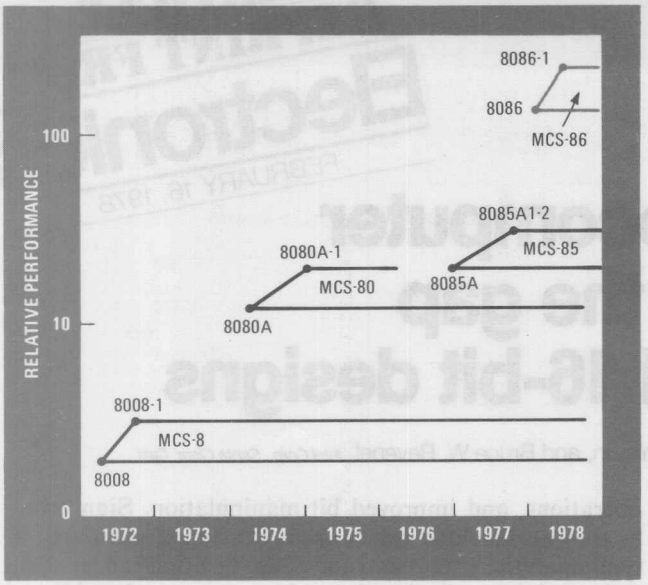
☐ The Intel 8086, a new microcomputer, extends the mid-range 8080 family into the 16-bit arena. The chip has attributes of both 8- and 16-bit processors. By executing the full set of 8080A/8085 8-bit instructions plus a powerful new set of 16-bit instructions, it enables a system designer familiar with existing 8080 devices to boost performance by a factor of as much as 10 while using essentially the same 8080 software package and development tools (Fig. 1).

The goals of the 8086 architectural design were to extend existing 8080 features symmetrically, across the board, and to add processing capabilities not to be found in the 8080. The added features include 16-bit arithmetic, signed 8- and 16-bit arithmetic (including multiply and divide), efficient interruptible byte-string operations, and improved bit manipulation. Significantly, they also include mechanisms for such minicomputer-type operations as reentrant code, position-independent code, and dynamically relocatable programs. In addition, the processor may directly address up to 1 megabyte of memory and has been designed to support multiple-processor configurations.

How it is done

The 8086's improved performance stems from a combination of process and architectural enhancements. It is the first microcomputer to be fabricated with the newly developed silicon-gate H-MOS process [*Electronics*, Aug. 18, 1977, p. 91], which gives the device 4-micrometer scaled-down metal-oxide-semiconductor





1. Next generation. Thanks to a faster MOS process and a generous number of powerful 16-bit instructions, the new 8086 attains a level of performance that is an order of magnitude higher than that of medium-range 8080-type microcomputers.

transistors and on-chip biasing that make it operate faster and more reliably.

The chip's general characteristics are listed in the table. With this high-performance MOS technique, typical on-chip gate propagation delays of 2 nanoseconds are as short as those obtained from costly Schottky transistor-transistor logic. This results in extremely fast internal clocking rates: 5 megahertz (200 ns) on standard chips and 8 MHz (125 ns) on selected chips. That is faster than any one-chip central processing unit now available. Since four CPU clock cycles correspond to approximately one memory cycle, the 8086 is much more efficient in accessing memory. Indeed, memory chip selection for the 8086 requires devices that cycle in 500 ns to 800 ns and access data (address to data-in valid) in a matter of 295 to 460 ns.

The H-MOS process also produces denser circuitry. The entire 16-bit data and microprogrammed control structures use 29,000 transistors integrated onto a die about 225 mils square. Many less complex peripheral chips that use large-scale integration are larger. The smallness of its die means that the high-performance 8086 will decline in cost as production experience with it grows, just as happened with the 8080, 8080A, and 8085.

An enhanced architecture

The architectural enhancements of the 8086 stem from a powerful register structure, increased memory address capability, almost unlimited levels of interrupts, and powerful input and output interface circuitry.

Unlike the 8080/8085 CPUs, the 8086's registers can process 16-bit as well as 8-bit data. The block diagram of Fig. 2 shows two register files. The general register file handles the 16-bit arithmetic/logic unit's instruction routines. It contains four 16-bit general data registers that are also addressable as 8-bit registers, two 16-bit memory base pointer registers, and two 16-bit index registers. All data manipulation instructions apply to all registers; certain addressing modes imply specific registers. All told, twice as many general-purpose data registers are provided as on 8-bit CPU chips. Complex arithmetical capability, very flexible memory addressing, and high computational throughput are the result.

The second bank of registers is designated the segment register file and extends the chip's addressing capabilities. It can address 1 megabyte of memory, in contrast to the 65,536-byte capacity of the 8080/8085. In this file, address relocation values for up to four 65-kilobyte program or data segments are stored in four 16-bit segment registers. The chip can control a full 65 kilobytes of address space for input/output ports.

Control and timing

The control and timing functions of the chip are extensive. A reset sequence control automatically establishes a clean machine environment, where during an interrupt much of the software overhead is performed by the CPU hardware. A multiprocessor lock facility enables the 8086 to share resources with a number of CPUs and peripheral controllers in large multiprocessor configurations. More important, though, are the independently controlled bus-interface unit and execution unit (Fig. 3).

The bus-interface unit maintains an optimized 6-byte fetch-ahead instruction queue. The queue is a technique found in larger computers for overlapping the execution and fetching of instructions. The bus-interface unit keeps the memory busy fetching words from the instruction stream, provided other system elements are not using memory cycles. The execution unit extracts bytes from the queue and executes them fast enough to eliminate dead time between instruction fetches. As many as two single-byte 8086 instructions can be executed within the time for one memory cycle in a fully overlapped fashion. The net effect is much improved performance for the same system bus bandwidth and memory speed.

Since the execution unit functions in parallel with the bus-interface unit's activities, it is largely unaffected by direct-memory-access operations, which also occur on the system bus—except, of course, when it requires a memory cycle for data itself.

Simulations of a wide variety of benchmark programs have indicated that six bytes is the optimum length for this instruction queue. Typical 8086 programs average two bytes per instruction, and their timing is somewhat execution-unit-bound—or to put it another way, the queue is usually nearly full. A smaller queue would be self-defeating, since it tends to make the timing instruction-fetch-bound. A larger queue, on the other hand, would aggravate the effect of wasted fetches of unused queue bytes when program branches occur.

Again, the 8086 processor performs 16-bit arithmetic, so the address objects that it manipulates in memory are 16 bits long. Since a 16-bit quantity can address only 65 kilobytes, additional mechanisms are required to build enough addresses to handle a 1-megabyte memory space.

This is done as shown in Fig. 4, where the 8086 memory is treated as an arbitrary number of segments, each at most 65 kilobytes in size. Each segment begins at an address that is evenly divisible by 16—the low-order 4

SUMMARY OF THE 8086'S GENERAL CHARACTERISTICS		
Process	H-MOS, scaled n-channel depletion-load silicon-gate (same as 2147 RAM)	
Transistors	29,000	
Package	40-lead Cerdip	
Supplies	5-volt, ground	
	Standard	Selected
Clock frequency	5 MHz	8 MHz
Memory cycle time (4 clocks/cycle)	800 ns	500 ns
Access time at pins (address to data-in valid)	460 ns	295 ns

bits of a segment's address are 0s. At any given moment, therefore, the contents of four of these segments are immediately addressable.

The four addressable memory segments are called the current code segment, the current data segment, the current stack segment, and the current extra segment. These segments need not be unique and indeed may overlap. The high-order 16 bits of the address of each current segment, called the segment address, is held in one of the four dedicated 16-bit segment registers (see Fig. 2). Bytes or words within a segment are addressed using 16-bit offset addresses within the 65-kilobyte segment (Fig. 5). A 20-bit physical address is constructed by adding the 16-bit offset address to the 16-bit segment address, complete with its four low-order zero bits.

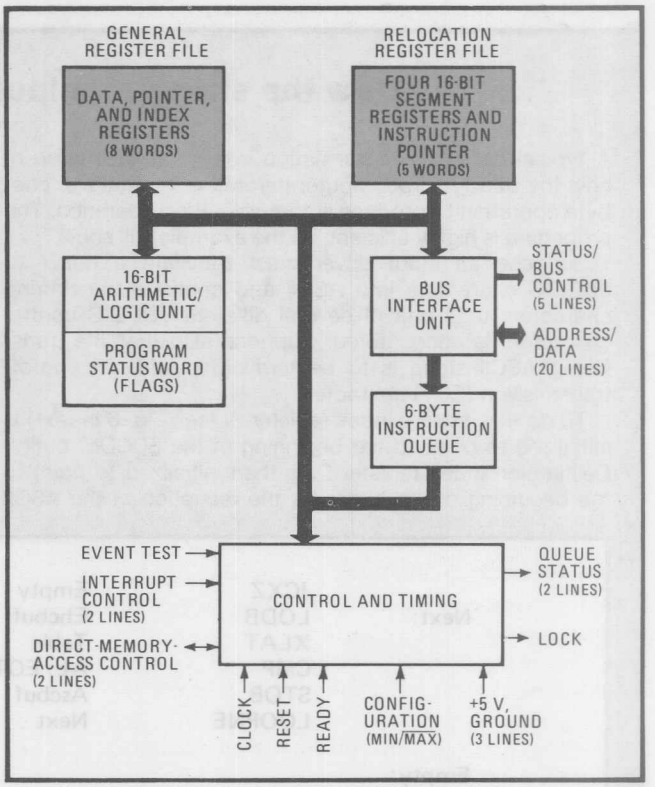
Of course, some programs may be designed not to load or manipulate the segment registers, and they are said to be dynamically relocatable. Such a program may be interrupted, moved in memory to a new location, and restarted with new segment register values.

The instruction set

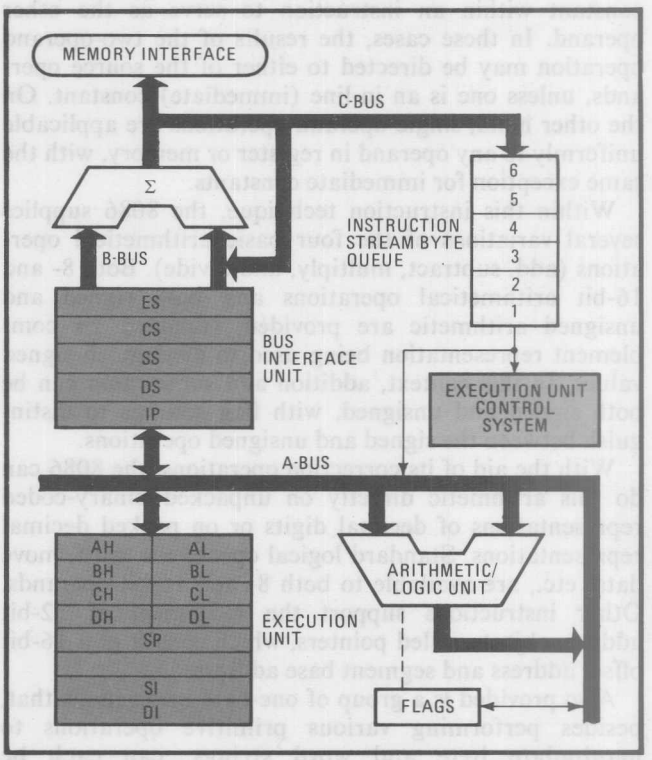
Several 8086 instructions may manipulate the four segment registers that make up the relocation register file. Most of the rest of the 8086's instruction set operates with the general register file, which contains two main sets of four 16-bit registers, as well as the 16-bit instruction pointer and the pair of 8-bit status flag registers (Fig. 6). The accumulator, base, counter, and data registers make up one set of general registers, the pointers and index registers the other set. The 8080 register set (shown tinted) is a subset of this structure.

The 8086 instruction set can address operands in several different ways. In general, operands in memory may be addressed either directly, with the 16-bit offset address, or indirectly, with base (BX or BP) and/or index (SI or DI) registers added to an optional 8- or 16-bit displacement constant. Here, if BX is used as the general-purpose data-base pointer, then the BP register may be used as a stack frame marker for efficiently supporting stack management in block-structured high-level languages like PLM-86. PLM-86 is an extension of the high-level PLM languages developed for the earlier 8080 and 8085 devices.

This two-operand technique allows memory or any



2. More processing power. The 8086 register structure is 16 bits wide internally. Additional pin functions can be supplied by time-multiplexing some pins (such as the address and data pins) and by strapping others to change the signal meaning.



3. Processing data. The 8086 has two independently controlled units: the bus-interface unit (BIU) maintains a fetch-ahead queue of instructions, which the execution unit (EU) performs. Throughput is enhanced because the BIU keeps the memory as busy as possible.

How the string-manipulation primitives operate

A typical data-format translation is a good illustration of how the 8086 microcomputer iterates with primitive, one-byte operation to produce a complex string operation. The procedure is highly efficient, as the example will show.

Suppose an input driver must translate a buffer of EBCDIC characters into ASCII and continue transferring characters until one of several different EBCDIC control characters is encountered. Suppose also that the transferred ASCII string is to be terminated with an end-of-transmission (EOT) character.

To do this, source index register SI (see Fig. 6 in text) is initialized to point to the beginning of the EBCDIC buffer. Destination index register DI is then initialized to point to the beginning of the buffer for the reception of the ASCII characters. Next, base register BX is made to point to an EBCDIC-into-ASCII translation data table residing in memory, and count register CX is initialized to contain the length of the EBCDIC buffer (which might possibly be empty). The translation table, incidentally, should contain the ASCII equivalent for each EBCDIC character plus perhaps ASCII nulls for illegal characters. The EOT code is then inserted among those entries in the table, wherever it corresponds to the desired EBCDIC stop character.

The table shows the 8086 instruction sequence that will implement these events. It makes the efficiency of the 8086 in performing such complex operations quite obvious: the entire body of this loop requires only seven bytes of code.

```
        JCXZ    Empty       ;skip if input buffer empty
Next:   LODB    Ebcbuf      ;fetch next EBCDIC character
        XLAT    Table       ;translate it to ASCII
        CMP     AL, EOT     ;test for the EOT
        STOB    Ascbuf      ;transfer ASCII character
        LOOPNE  Next        ;continue if not EOT

Empty:
```

register to serve as one operand and either a register or a constant within an instruction to serve as the other operand. In these cases, the results of the two-operand operation may be directed to either of the source operands, unless one is an in-line (immediate) constant. On the other hand, single-operand operations are applicable uniformly to any operand in register or memory, with the same exception for immediate constants.

Within this instruction technique, the 8086 supplies several variations of the four basic arithmetical operations (add, subtract, multiply, and divide). Both 8- and 16-bit arithmetical operations and both signed and unsigned arithmetic are provided, standard 2's complement representation being used to distinguish signed values. In this context, addition and subtraction can be both signed and unsigned, with flag settings to distinguish between the signed and unsigned operations.

With the aid of its correction operations, the 8086 can do this arithmetic directly on unpacked binary-coded representations of decimal digits or on packed decimal representations. Standard logical operations, shift, move data, etc., are available to both 8- and 16-bit operands. Other instructions support the movement of 32-bit address objects called pointers, which consist of a 16-bit offset address and segment base address.

Also provided is a group of one-byte instructions that, besides performing various primitive operations to manipulate byte and word strings, can each be performed repeatedly when provided with a special prefix. The single-operation forms are then combined to form complex strings of operations, with their repetition controlled by special iterative operations. The effect is to create tight, efficient loops for performing subroutines (see "How the string-manipulation primitives operate," above).

For handling program flow, two basic varieties of calls, jumps, and returns are provided—one that transfers control within the current code segment, and one that transfers control to an arbitrary code segment, which then becomes the current code segment. The 8086 supports direct and indirect transfers, both of which make use of the standard addressing modes. Intra-segment calls and jumps specify a self-relative displacement, thus allowing position-independent code.

In all, 16 conditional jumps are provided to support all common program-controlled interrupt structures. Both signed and unsigned relationships can be tested, as well as parity, overflow, and sign conditions.

Many interrupt types

In an interrupt sequence, an interrupt signal prompts the transfer of processor control to a new location in a new code segment. The 8086 memory structure supplies a 256-element table that contains pointers to these interrupt service code locations. Each element is four bytes in size, containing an offset address and a segment address for the service code location. Each element of this table corresponds to an interrupt type, and there are 256 interrupt types, or enough for even the most heavily interrupt-driven applications.

The 8086 has the ability to detect program inconsistencies, such as errors in division or overflow conditions, by means of a one-byte instruction that causes an interrupt if the condition occurs. The internal interrupt

instructions transfer the program's execution control to a checking sequence by means of operations like those done during external interrupts. Both the internal and external interrupts perform a program transfer by pushing the flag register onto the stack and then making an indirect call (of the intersegment variety) to the service routine.

Multiprocessing mechanisms

Besides handling a large variety of interrupts, the 8086 CPU has mechanisms for sharing resources and controlling access to those resources in multiprocessor applications. Such mechanisms are mostly provided by software operating systems but do require some hardware assistance. This is where the bus-locked output comes into use.

Labeled lock in Fig. 2, the output works like this. The 8086 has a special one-byte prefix that can be attached to the front of any instruction. This prefix then compels the processor to assert a bus-lock signal for the duration of the operation caused by that instruction. Meanwhile, external hardware, upon receipt of that same signal, is prohibiting other bus masters from bus access during the period of its assertion.

The instruction most likely to have such a prefix attached to it is "exchange register with memory." A simple software lock may be implemented with the following code sequence:

```
Check:  MOV   AL,1        ;set AL to 1
                          (implies locked)
LOCK    XCHG  Sema, AL    ;test and set lock
        TEST  AL, AL      ;set flags based on AL
        JNZ   Check       ;retry if lock already set
                          ;critical region
        MOV   Sema,0      ;clear the lock when done
```

System configurations

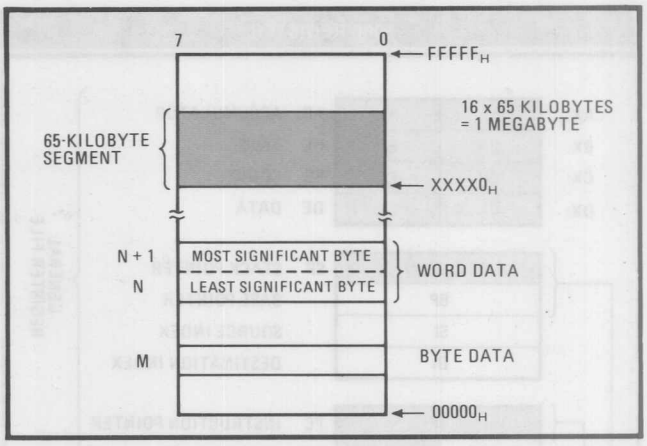
A wide variety of system configurations can be built around the 8086 CPU. For small systems, where minimal external circuitry is needed, the 8086 may be strapped into a minimum mode, in which the microprocessor itself provides all bus control signals. In larger or Multibus systems, the strap pin may be set to the maximum mode, and the same control-signal pins take on the functions required to operate the 8288 bus controller.

For really big systems, the MCS 86 family of components includes several new Schottky bipolar support components that maximize the CPU's ability to control many external memory and peripheral chips. There are inverting and noninverting octal latches and octal bus transceivers, a clock controller/ready synchronizer, and a bus controller that makes the 8086 compatible with the 8080 family's Multibus timing and control protocols.

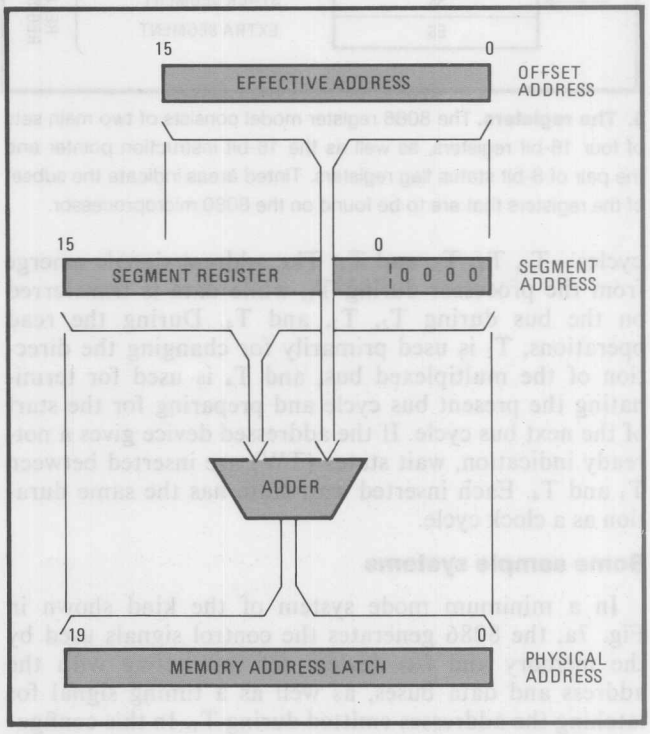
The latches and transceivers have three-state controls and separate strobe or direction signals. To ease system design, they are packaged in 20-pin packages with a uniform pinout. The bus side of the parts is capable of sinking 32 milliamperes of output current for driving a wide assortment of peripheral equipment.

The clock chip includes an oscillator circuit, a Schmitt-trigger-reset detector/driver, and a ready-synchronizer circuit. The chip's latch, which captures the asynchronous ready signal from the system or Multibus, prevents output glitches or metastable conditions in presenting the signal to the CPU within the required timing constraints. The bus controller, used when the CPU is strapped to maximum mode, generates Multibus command signals as well as control signals for the address latches and data transceivers that are used in buffered systems.

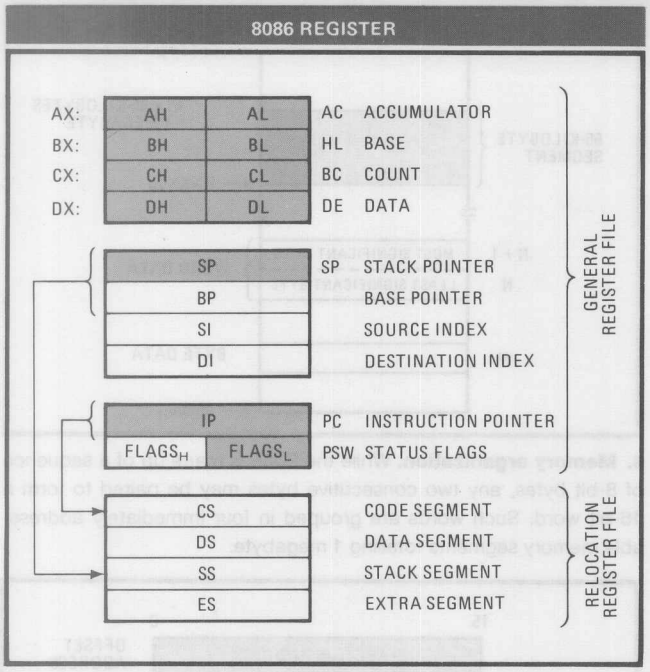
The multiplexing technique used by the 8086 CPU for address and data resembles the one used on the 8085. Each processor bus cycle consists of at least four clock



4. Memory organization. While the 8086 is made up of a sequence of 8-bit bytes, any two consecutive bytes may be paired to form a 16-bit word. Such words are grouped in four immediately addressable memory segments totalling 1 megabyte.



5. Addressing it. The 8086 memory can be thought of as an arbitrary number of segments. A byte or word within a segment is addressed with a 16-bit offset address. Adding the offset address to the 16-bit segment address creates a 20-bit physical address.



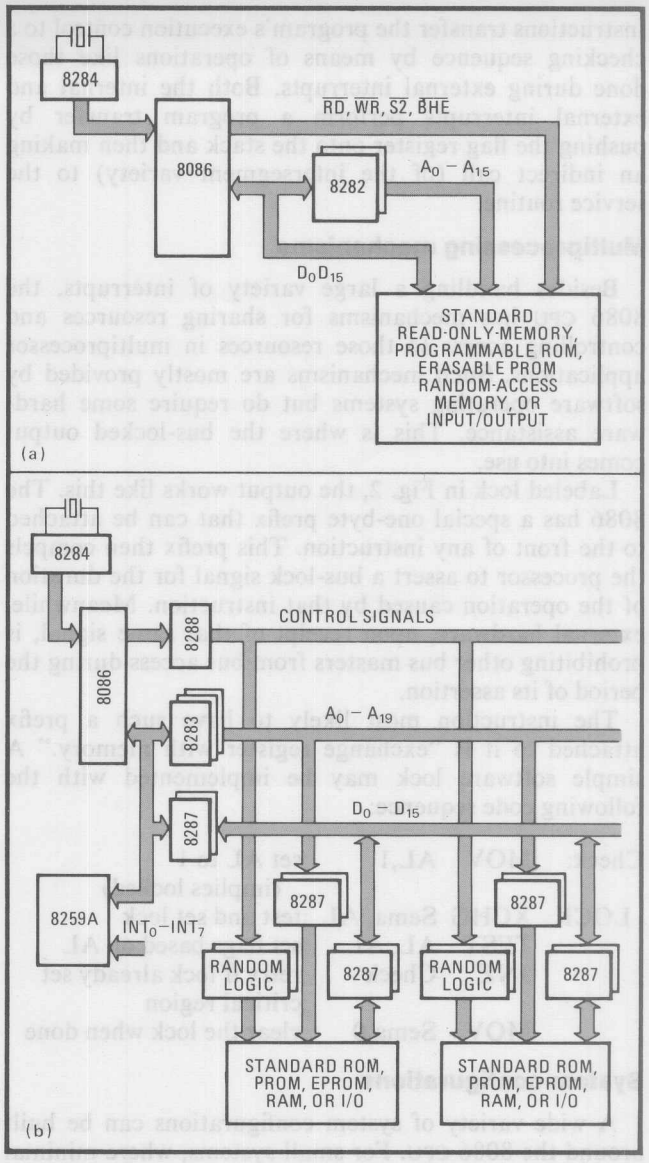
6. The registers. The 8086 register model consists of two main sets of four 16-bit registers, as well as the 16-bit instruction pointer and the pair of 8-bit status flag registers. Tinted areas indicate the subset of the registers that are to be found on the 8080 microprocessor.

cycles—T_1, T_2, T_3, and T_4. The address signals emerge from the processor during T_1, while data is transferred on the bus during T_2, T_3, and T_4. During the read operations, T_2 is used primarily for changing the direction of the multiplexed bus, and T_4 is used for terminating the present bus cycle and preparing for the start of the next bus cycle. If the addressed device gives a not-ready indication, wait states (TW) are inserted between T_3 and T_4. Each inserted wait state has the same duration as a clock cycle.

Some sample systems

In a minimum mode system of the kind shown in Fig. 7a, the 8086 generates the control signals used by the memory and I/O devices for interacting with the address and data buses, as well as a timing signal for latching the addresses emitted during T_1. In this configuration, as noted earlier, the access times required of memory and I/O devices for the 5-MHz 8086 are roughly 430 ns from receipt of address and 205 ns from receipt of read or write enable. These times are about 265 ns and 130 ns respectively for 8-MHz CPU operation. The difference between these figures and those in the table takes the addition of the address latch into account.

In the larger buffered configuration (Fig. 7b), the 8086, in its maximum mode, generates coded status information during T_1 on only some of the pins needed for producing minimum-mode control signals. Here, all that is needed is enough coded information to push an 8288 bus controller into generating Multibus-compatible control signals and timing signals for addressing latches and data transceivers. The minimum-mode control-signal pins can now take on additional functions, such as extra direct-memory-access control and bus-locking



7. Systems. When operating in the minimum mode, the 8086 needs only 11 components to form a complete system, including clock, 2 kilobytes of RAM, and 4 kilobytes of ROM (a). By adding support components, very large systems (b) can be configured.

capabilities for use in multiprocessor operations.

In buffered systems the required memory and I/O access times for 5-MHz CPU operation are 395 ns from receipt of address and 290 ns from receipt of read command. These times are respectively about 230 ns and 150 ns for 8-MHz CPU operation.

The bottom line of all system design is performance. Initial studies have indicated that on average an 8086-based system will perform about an order of magnitude better than an 8080A-based one. Depending on program type, execution speeds 7 to 12 times faster than 8080A speeds can be expected. At the same time, the program is typically 10% to 25% shorter. However, while there exists an 8086 instruction mapping for every 8080 instruction, and while 8080 programs may be easily transferred to the 8086, maximum 8086 efficiency does require the rewriting of certain routines. □